



TITLE:

# Parallel Algorithms for the Maximal Tree Cover Problems

AUTHOR(S):

CHEN, Zhi-Zhong

---

CITATION:

CHEN, Zhi-Zhong. Parallel Algorithms for the Maximal Tree Cover Problems. 数理解析研究所講究録 1992, 796: 43-53

ISSUE DATE:

1992-07

URL:

<http://hdl.handle.net/2433/82762>

RIGHT:

## Parallel Algorithms for the Maximal Tree Cover Problems

Zhi-Zhong CHEN

*Department of Computer Science and Information Mathematics,  
University of Electro-Communications,  
1-5-1 Chofugoka, Chofu-shi, Tokyo 182.*

**Abstract** A maximal  $l$ -diameter tree cover of a graph  $G = (V, E)$  is a spanning subgraph  $C = (V, E_C)$  of  $G$  such that each connected component of  $C$  is a tree,  $C$  contains no path with more than  $l$  edges, and adding any edge in  $E - E_C$  to  $C$  yields either a path of length  $l + 1$  or a cycle. For every function  $f$  from positive integers to positive integers, the maximal  $f$ -diameter tree cover problem (MDTC( $f$ ) problem for short) is to find a maximal  $f(n)$ -diameter tree cover of  $G$ , given an  $n$ -node graph  $G$ . In this paper, we give three parallel algorithms for the MDTC( $f$ ) problem. The first algorithm can be implemented in time  $O(T_{MSP}(n, f(n)) + \log^2 n)$  using polynomial number of processors on a P-RAM, where  $T_{MSP}(n, f(n))$  is the time needed to find a maximal set of vertex disjoint paths of length  $f(n)$  in a given  $n$ -node graph using polynomial number of processors on a P-RAM. We then show that if suitable restrictions are imposed on the input graph and/or on the magnitude of  $f$ , then  $T_{MSP}(n, f(n)) = O(\log^k n)$  for some constant  $k$  and thus, for such cases, we obtain an NC algorithm for the MDTC( $f$ ) problem. The second algorithm runs in time  $O(\frac{n \log^2 n}{f(n)+1})$  using polynomial number of processors on a P-RAM. Thus if  $f(n) = \Omega(\frac{n}{\log^k n})$  for some  $k \geq 0$ , we obtain an NC algorithm for the MDTC( $f$ ) problem. The third algorithm is a randomized one and can be implemented in time  $O(\log^6 n)$  using polynomial number of processors on a P-RAM for arbitrary functions and graphs.

## 1 Introduction

Parallel algorithms for specific maximal subgraph problems and their natural extensions have received substantial attention recently [1, 2, 9, 10, 11, 12, 13, 17, 18]. Two outstanding maximal subgraph problems are the maximal independent set (MIS) problem and the maximal matching (MM) problem. Much work has been done on the development of efficient parallel algorithms for these two problems [1, 9, 10, 11, 12, 13]. As natural extensions of the MIS problem, there have been the maximal bipartite set problem [16] and the bounded degree maximal subgraph problem [17]. However to our knowledge, no natural extension of the MM problem is known. In this paper, we give a natural extension of the MM problem and present three parallel algorithms for solving it.

A *tree cover* of a graph  $G$  is a spanning subgraph of  $G$  in which each connected component is a tree. An  $l$ -*diameter tree cover* of a graph  $G$  is a tree cover of  $G$  that contains no path with more than  $l$  edges. An  $l$ -diameter tree cover  $C = (V, E_C)$  of a graph  $G = (V, E)$  is said to be *maximal* if for each edge  $e \in E - E_C$ , the graph  $(V, E_C \cup \{e\})$  contains either a path of length  $l + 1$  or a cycle. For every function  $f$  from positive integers to positive integers, the *maximal  $f$ -diameter tree cover problem* (the MDTC( $f$ ) problem for short) is to find a maximal  $f(n)$ -diameter tree cover, given an  $n$ -node graph  $G$ . By this definition, the MM problem can be viewed as the MDTC( $f$ ) problem where  $f(n) = 1$  for each  $n$ , and the MDTC( $f$ ) problem becomes the problem of finding a spanning forest if  $f$  is the identity function. Thus we can view the MDTC( $f$ ) problem as a natural extension of both problems above.

We are interested in designing efficient parallel algorithms for the MDTC( $f$ ) problem. If computing  $f$  is hard, say is hard for PTIME, then the MDTC( $f$ ) problem may not have an efficient parallel algorithm.

This says that it is necessary to give some assumption on the complexity of  $f$  in order to parallelize the  $\text{MDTC}(f)$  problem. Hence we shall assume that  $f$  is computable in  $\text{NC}^2$ . Under this assumption, two parallel algorithms are presented for the  $\text{MDTC}(f)$  problem. The first algorithm can be implemented in time  $O(T_{\text{MSP}}(n, f(n)) + \log^2 n)$  using polynomial number of processors on a P-RAM, where  $T_{\text{MSP}}(n, f(n))$  is the time needed to find a maximal set of vertex disjoint paths of length  $f(n)$  in a given  $n$ -node graph using polynomial number of processors on a P-RAM. In general, it seems unlikely that  $T_{\text{MSP}}(n, f(n)) = O(\log^k n)$  for some constant  $k$ , because computing a maximal set of vertex disjoint path of length  $n - 1$  in an  $n$ -node graph is equivalent to computing a Hamiltonian path in the graph. However, if one of the following (1), (2), and (3) holds, then it can be shown that  $T_{\text{MSP}}(n, f(n)) = O(\log^k n)$  for some constant  $k$ , and thus, NC algorithms can be obtained for the three cases: (1)  $f$  is a constant function, i.e.,  $f$  maps each integer to a fixed constant; (2)  $f(n) = O(\log n)$  and the input graph is of bounded degree; (3) the input graph is either a tree or an  $n$ -node graph with minimum degree at least  $\frac{n}{2}$ . Our second algorithm runs in time  $O(\frac{n \log^2 n}{f(n)+1})$  using polynomial number of processors on a P-RAM. Thus if  $f(n) = \Omega(\frac{n}{\log^k n})$  for some  $k \geq 0$ , we obtain an NC algorithm for the  $\text{MDTC}(f)$  problem. Our third algorithm is a randomized one and can be implemented in time  $O(\log^6 n)$  using polynomial number of processors on a P-RAM. The basic idea used in the algorithms is to extend a simple path of length  $f(n)$  in the graph to a subtree maximal with respect to the condition that the subtree contains no simple path with more than  $f(n)$  edges. The key idea used in the third algorithm is that of path separators [3, 4].

## 2 Preliminaries

Throughout this paper, we mean, by a graph, an undirected graph without any multiple edges and self-loops. A graph may be connected or not. Let  $G = (V, E)$  be a graph. We sometimes write  $V = V(G)$  and  $E = E(G)$ . For a subset  $U \subseteq V$ , the *subgraph of  $G$  induced by  $U$*  is the graph  $(U, F)$  with  $F = \{\{u, v\} \in E : u, v \in U\}$ . Unless stated otherwise, by a path, we mean a simple path. The *length* of a path is the number of edges it traverses. We use  $|p|$  to denote the length of a path  $p$ . Two paths are *vertex disjoint* if they share no common vertex. We often identify a set  $P$  of paths with the graph consisting of vertices and edges on paths of  $P$ , and hence  $V(P)$  and  $E(P)$  mean the sets of all vertices and edges on paths of  $P$ , respectively. If  $P$  is a set containing a single path  $p$ , then we identify  $P$  with  $p$ . A *tree cover* of a graph is a spanning subgraph in which each connected component is a tree. An  *$l$ -diameter tree cover* of a graph  $G$  is a tree cover  $C$  of  $G$  that contains no path of length  $l + 1$ . An  *$l$ -diameter tree cover  $C = (V, E_C)$  of a graph  $G = (V, E)$  is said to be maximal* if for each edge  $e \in E - E_C$ , the graph  $(V, E_C \cup \{e\})$  contains a path of length  $l + 1$  or a cycle. We denote by  $\text{dist}_G(u, v)$  the distance between two vertices  $u, v$  in a graph  $G$ , and denote by  $G_1 \cup G_2$  the graph  $(V_1 \cup V_2, E_1 \cup E_2)$ , where  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$ . By a *function*, we mean a function from positive integers to positive integers. Unless stated otherwise, all functions in this paper are assumed to be computable in  $\text{NC}^2$ .

For every function  $f$ , the *maximal  $f$ -diameter tree cover problem* (the  $\text{MDTC}(f)$  problem for short) is defined by

**Instance:** An  $n$ -node graph  $G$ .

**Problem:** Find a maximal  $f(n)$ -diameter tree cover of  $G$ .

### 3 A basic procedure

In this section, we give a basic procedure that will be used in our algorithms for the MDTC( $f$ ) problem. We assume that all vertices in an  $n$ -node graph are linearly ordered by indexing them with numbers between 1 through  $n$ . The procedure has the following description.

**Procedure** *Extend*( $G_{in}, P$ )

*Input:* A graph  $G_{in} = (V_{in}, E_{in})$  and a set  $P = \{p_1, \dots, p_k\}$  of vertex disjoint paths of length  $l$  in  $G_{in}$ .

*Output:* A subgraph  $G_{out} = (V_{out}, E_{out})$  of  $G_{in}$  such that  $G_{out}$  contains all vertices in  $V(P)$  but contains neither a path of length  $l + 1$  nor a cycle, and for each edge  $\{u, v\} \in E_{in} - E_{out}$  with  $u \in V_{out}$  or  $v \in V_{out}$ , the graph  $(V_{out} \cup \{u, v\}, E_{out} \cup \{\{u, v\}\})$  contains a path of length  $l + 1$  or a cycle.

*Initialization:* Set  $V_{out}$  to  $V(P)$  and set  $E_{out}$  to  $E(P)$ .

**begin**

1. Compute  $H = (V_{in}, E_{in} - \{\{v, v'\} \in E_{in} : v, v' \in V(P)\})$ .
2. In parallel, for each vertex  $v \in V(P)$ , compute  $pos(v, P) = \max\{dist_P(v_1, v), dist_P(v_2, v)\}$ , where  $v_1$  and  $v_2$  are two endpoints of the path in  $P$  containing  $v$ . (Note:  $\lceil \frac{l}{2} \rceil \leq pos(v, P) \leq l$ .)
3. In parallel, for each vertex  $v \in V(P)$ , perform the following two steps:
  - 3.1 Compute  $N(v, P) = \{u \in V_{in} - V(P) : v \text{ is the vertex in } V(P) \text{ with the smallest index such that } pos(v, P) + dist_H(v, u) \leq l \text{ and } pos(v, P) + dist_H(v, u) \leq pos(v', P) + dist_H(v', u) \text{ for all } v' \in V(P)\}$ .
  - 3.2 Compute a breadth-first spanning tree rooted at  $v$  of the subgraph of  $H$  induced by  $\{v\} \cup N(v, P)$ , and add the tree to  $G_{out}$ .

**end**

We below show the correctness and the running time of procedure *Extend*. The notations in the procedure are used in the remainder of this section. By the procedure, we immediately have three useful facts:

**Fact 1** The subgraph of  $H$  induced by  $\{v\} \cup N(v, P)$  is connected for every  $v \in V(P)$ , and  $N(v', P) \cap N(v'', P) = \emptyset$  for every two distinct vertices  $v'$  and  $v''$  in  $V(P)$ .

*Proof.* It is obvious from the definition of  $N(v, P)$  that there is a path in  $H$  with no more than  $l$  edges between  $v$  and every vertex of  $N(v, P)$ . Thus we have the first assertion. If a vertex  $u$  is in  $N(v, P)$  for some  $v \in V(P)$ , then  $v$  is of the smallest index among all such vertices satisfying the condition on  $u$  and  $v$  in the definition of  $N(v, P)$ . Thus the second assertion is obvious. ■

**Fact 2** Let  $v$  be a vertex in  $V(P)$ , and let  $u$  be a vertex in  $N(v, P)$ . Then the distance between  $v$  and  $u$  in any breadth-first spanning tree of the subgraph of  $H$  induced by  $\{v\} \cup N(v, P)$  is equal to  $dist_H(v, u)$ .

*Proof.* This follows from a well-known fact that if  $T$  is a breadth-first spanning tree rooted at  $v$  of a connected graph  $G$  and  $u$  is a vertex in  $T$ , then the distance between  $v$  and  $u$  in  $T$  is equal to the distance between  $u$  and  $v$  in  $G$  [8]. ■

**Fact 3** Let  $u \in V_{in} - V(P)$ . Then,  $u$  is contained in the output of *Extend*( $G_{in}, P$ ) if and only if there exists a vertex  $v \in V(P)$  such that  $pos(v, P) + dist_H(v, u) \leq l$ .

**Lemma 3.1**  $G_{out}$  contains neither a path of length  $l + 1$  nor a cycle.

*Proof.* By the procedure and Fact 1, it is easy to see that  $G_{out}$  contains no cycle. What we need to show is that  $G_{out}$  contains no path of length  $l + 1$ . It is sufficient to show that each connected component  $T$  of  $G_{out}$  contains no path of length  $l + 1$ . By the procedure and Fact 1, we know that  $T$  must contain exactly one path  $p$  of  $P$ . Fix two arbitrary vertices  $w_1$  and  $w_2$  in  $T$ . Then we distinguish three cases below.

*Case 1: both  $w_1$  and  $w_2$  are on  $p$ .* Since  $p$  has length  $l$ ,  $\text{dist}_T(w_1, w_2) \leq l$ .

*Case 2: there exists a vertex  $v$  on  $p$  such that  $w_1$  and  $w_2$  are contained in  $\{v\} \cup N(v, P)$ .* By the definition of  $N(v, P)$ , we know that  $\text{pos}(v, P) + \text{dist}_H(v, w_1) \leq l$  and that  $\text{pos}(v, P) + \text{dist}_H(v, w_2) \leq l$ . Since  $\text{pos}(v, P)$  is no less than  $\lceil \frac{l}{2} \rceil$ , we further know that  $\text{dist}_H(v, w_1) \leq \lfloor \frac{l}{2} \rfloor$  and that  $\text{dist}_H(v, w_2) \leq \lfloor \frac{l}{2} \rfloor$ . Combining these observations with Fact 2, we have that  $\text{dist}_T(v, w_1) \leq \lfloor \frac{l}{2} \rfloor$  and that  $\text{dist}_T(v, w_2) \leq \lfloor \frac{l}{2} \rfloor$ . Hence  $\text{dist}_T(w_1, w_2) \leq \text{dist}_T(w_1, v) + \text{dist}_T(w_2, v) \leq l$ .

*Case 3: there exist two distinct vertices  $v'$  and  $v''$  on  $p$  such that  $w_1 \in \{v'\} \cup N(v', P)$  and  $w_2 \in \{v''\} \cup N(v'', P)$ .* Let  $v_1$  and  $v_2$  be  $p$ 's two endpoints. Noting that  $\text{dist}_T(v', v_i) = \text{dist}_P(v', v_i)$  for  $1 \leq i \leq 2$ , we have that  $\text{dist}_T(v_1, v') + \text{dist}_T(v', v_2) = l$  and that  $\text{dist}_T(v', w_1) + \text{pos}(v', P) = \text{dist}_T(v', w_1) + \max\{\text{dist}_T(v_1, v'), \text{dist}_T(v', v_2)\}$ . Since  $w_1 \in \{v'\} \cup N(v', P)$ , we know that  $\text{dist}_T(v', w_1) + \text{pos}(v', P) \leq l$ . From these facts, we easily see that  $\text{dist}_T(v', w_1) \leq \min\{\text{dist}_T(v_1, v'), \text{dist}_T(v', v_2)\}$ . Similarly, we can see that  $\text{dist}_T(v'', w_2) \leq \min\{\text{dist}_T(v_1, v''), \text{dist}_T(v'', v_2)\}$ . Thus, we have:

$$\begin{aligned} \text{dist}_T(w_1, w_2) &= \text{dist}_T(w_1, v') + \text{dist}_T(v', v'') + \text{dist}_T(v'', w_2) \\ &\leq \min\{\text{dist}_T(v_1, v'), \text{dist}_T(v', v_2)\} + \text{dist}_T(v', v'') + \min\{\text{dist}_T(v_1, v''), \text{dist}_T(v'', v_2)\} \\ &\leq l. \end{aligned}$$

**Lemma 3.2** For every edge  $\{w_1, w_2\} \in E_{in} - E_{out}$  with  $w_1 \in V_{out}$  or  $w_2 \in V_{out}$ , the graph  $(V_{out} \cup \{w_1, w_2\}, E_{out} \cup \{\{w_1, w_2\}\})$  contains a path of length  $l + 1$  or a cycle.

*Proof.* There are two cases that can occur:

*Case 1: both  $w_1$  and  $w_2$  are contained in  $G_{out}$ .* Note that if some connected component of  $G_{out}$  contains both  $w_1$  and  $w_2$ , then the graph  $(V_{out} \cup \{w_1, w_2\}, E_{out} \cup \{\{w_1, w_2\}\})$  contains a cycle. Thus, we may assume that some connected component  $T_1$  of  $G_{out}$  contains  $w_1$  and another connected component  $T_2$  of  $G_{out}$  contains  $w_2$ . By the procedure, we know that each of  $T_1$  and  $T_2$  must contain a path of  $P$ . Let  $p_1$  and  $p_2$  be the paths of  $P$  contained in  $T_1$  and  $T_2$ , respectively. Let  $v_1$  and  $v_2$  be  $p_1$ 's two endpoints, and let  $v_3$  and  $v_4$  be  $p_2$ 's two endpoints. Since  $\text{dist}_{T_1}(v_1, v_2) = l$  and  $T_1$  is a tree, it is obvious that  $\max\{\text{dist}_{T_1}(w_1, v_1), \text{dist}_{T_1}(w_1, v_2)\} \geq \lceil \frac{l}{2} \rceil$ . Similarly, it holds that  $\max\{\text{dist}_{T_2}(v_3, w_2), \text{dist}_{T_2}(v_4, w_2)\} \geq \lceil \frac{l}{2} \rceil$ . These facts imply that when edge  $\{w_1, w_2\}$  is added to  $G_{out}$ , the resulting graph has a path with more than  $l$  edges.

*Case 2: only one of  $w_1$  and  $w_2$  is contained in  $G_{out}$ .* Without loss of generality, we may assume that  $w_1$  is contained in  $G_{out}$  while  $w_2$  is not. Let  $v$  be the vertex in  $V(P)$  such that  $w_1 \in N(v, P) \cup \{v\}$ . Then, it holds that  $\text{pos}(v, P) + \text{dist}_H(v, w_1) = l$ , or else  $\text{pos}(v, P) + \text{dist}_H(v, w_2) \leq l$  which contradicts that  $w_2$  is not in  $G_{out}$  by Fact 3. Let  $v_1$  and  $v_2$  be the endpoints of the path of  $P$  containing  $v$ . We now have that  $\max\{\text{dist}_{G_{out}}(v_1, w_1), \text{dist}_{G_{out}}(v_2, w_1)\} = l$  by Fact 2 and the definition of  $\text{pos}(v, P)$ . This, in turn, implies that when edge  $\{w_1, w_2\}$  is added to  $G_{out}$ , the resulting graph contains a path of length  $l + 1$ .  $\blacksquare$

NC<sup>2</sup> algorithms are known for computing the distance between two vertices in a graph and for computing a breadth-first spanning tree of a connected graph [6]. Thus all steps of procedure *Extend* can be performed in time  $O(\log^2 n)$  using polynomial number of processors on a P-RAM. Hence we immediately have the following theorem by summarizing the results above.

**Theorem 3.1** Procedure *Extend* is correct and can be executed in  $O(\log^2 n)$  time using polynomial number of processors on a P-RAM, where  $n$  is the number of vertices in the input graph.

## 4 The first algorithm

In this section, we present our first parallel algorithm that finds a maximal  $f(n)$ -diameter tree cover of a given  $n$ -node graph. The algorithm unifies several disparate algorithms corresponding to several special cases which

will be discussed in the latter half of this section. Those disparate algorithms differ from each other only in the implementation of Stage 1 shown below. In the first half of this section, we first show the correctness of the algorithm and then show the running time of each stage except Stage 1 of the algorithm. The algorithm has the following description.

**Algorithm 1**

*Input:* An  $n$ -node graph  $G_1 = (V_1, E_1)$ .

*Stage 1:*

1. Compute a maximal set  $P$  of vertex disjoint paths of length  $f(n)$  in  $G_1$ .  
(*Note:* By maximal, we mean that when all vertices in  $P$  are removed from  $G_1$ , the resulting graph contains no path with  $f(n)$  edges.)

*Stage 2:*

2. Set  $C_1$  to the output of  $Extend(G_1, P)$ .

*Stage 3:*

3. Set  $G_2$  to the subgraph of  $G_1$  induced by  $V_1 - V(C_1)$  and set  $C_2$  to the empty graph.
4. In parallel, for each connected component of  $G_2$ , compute its spanning tree and add the tree to  $C_2$ .

*Output:*  $C_1 \cup C_2$ .

**End of Algorithm 1.**

We first show the correctness of Algorithm 1. In addition to the notations used in the algorithm, let  $C_{out} = C_1 \cup C_2$ , i.e.,  $C_{out}$  is the output of Algorithm 1.

**Lemma 4.1**  $C_{out}$  is a maximal  $f(n)$ -diameter tree cover of  $G_1$ .

*Proof.* It is easy to see that  $V(C_{out}) = V(G_1)$  and that the induced graph of each  $C_i$  contain neither a path with more than  $f(n)$  edges nor a cycle by the algorithm and Theorem 3.1. Noting that  $C_1$  and  $C_2$  share no common vertex, we have that  $C_{out}$  is an  $f(n)$ -diameter tree cover of  $G_1$ . Next, we show that for each edge  $e \in E_1 - E(C_{out})$ , adding  $e$  to  $C_{out}$  yields either a path of length  $f(n) + 1$  or a cycle. To show this, it suffices to show that for each  $i$  with  $1 \leq i \leq 2$  and each edge  $\{w_1, w_2\} \in E(G_i) - E(C_i)$  with  $w_1 \in V(C_i)$  or  $w_2 \in V(C_i)$ , the graph  $(V(C_i) \cup \{w_1, w_2\}, E(C_i) \cup \{\{w_1, w_2\}\})$  contains a path of length  $l + 1$  or a cycle. This, however, follows immediately from the algorithm, Theorem 3.1, and Fact 3 in Section 3. ■

It is easy to see that all steps of Algorithm 1 except step 1 of Stage 1 can be performed in time  $O(\log^2 n)$  using polynomial number of processors on a P-RAM. Hence we immediately have the following theorem.

**Theorem 4.1.** Let  $T_{MSP}(n, l)$  be the time needed to find a maximal set of vertex disjoint paths of length  $l$  in a given  $n$ -node graph using polynomial number of processors on a P-RAM. Then for every function  $f$ , Algorithm 1 outputs a maximal  $f(n)$ -diameter tree cover of a given  $n$ -node graph, and can be performed in time  $O(T_{MSP}(n, f(n)) + \log^2 n)$  using polynomial number of processors on a P-RAM.

In the remainder of this section, we consider the implementation of Stage 1 of Algorithm 1 given in the above. In general, it seems unlikely that NC implementations of Stage 1 exist, because computing a maximal set of vertex disjoint paths of length  $n - 1$  in an  $n$ -node graph is equivalent to computing a Hamiltonian path in the graph. Here we consider several special cases where the input graph and/or the magnitude of  $f$  are suitably restricted so that Stage 1 has NC implementations, and thus NC algorithms are obtained for the MDTC( $f$ ) problem in these special cases by Algorithm 1.

**Lemma 4.2** Given an  $n$ -node graph  $G$  and a positive integer  $l$ , finding a maximal set  $P$  of vertex disjoint paths of length  $l$  in  $G$  can be done in time  $O(\log^2 n)$  using polynomial number of processors on a P-RAM if

one of the following conditions holds: (1)  $l$  is a fixed constant; (2)  $l = O(\log n)$  and  $G$  is of bounded degree  $d$ ; (3)  $G$  is a tree.

*Proof.* To find  $P$ , we may first construct a graph  $H = (V_H, E_H)$  and then compute a maximal independent set in  $H$ , where

$$V_H = \{p : p \text{ is a path of length } l \text{ in } G\}, \text{ and}$$

$$E_H = \{\{p, p'\} : p \text{ and } p' \text{ are elements of } V_H \text{ and share a common vertex}\}.$$

To compute all elements of  $V_H$  and  $E_H$ , we may simply enumerate all paths of length  $l$  in  $G$ , and may check, for all pairs of those paths, whether they share a common vertex. The enumeration and checks can be done in time  $O(1)$ ,  $O(l)$ , and  $O(\log^2 n)$  using  $O(n^l)$ ,  $O(nd^l)$ , and  $O(n^2)$  processors on a P-RAM, respectively for the three cases (1)-(3). To find a maximal independent set in  $H$ , we may use the NC<sup>2</sup> algorithm for the MIS problem given by Luby [13]. Hence the lemma holds. ■

By combining Theorem 4.1 and Lemma 4.2, we immediately have the following corollary.

**Corollary 4.1** Given an  $n$ -node graph  $G_1$ , a maximal  $f(n)$ -diameter tree cover of  $G_1$  can be found in time  $O(\log^2 n)$  using polynomial number of processors on a P-RAM if one of the following conditions holds: (1)  $f(n) = c$  for some constant  $c$ ; (2)  $f(n) = O(\log n)$  and  $G_1$  is of bounded degree; (3)  $G_1$  is a tree.

**Lemma 4.3** Given an  $n$ -node graph  $G$  with minimum degree at least  $\frac{n}{2}$  and a positive integer  $l$ , finding a maximal set  $P$  of vertex disjoint paths of length  $l$  in  $G$  can be done in time  $O(\log^4 n)$  using polynomial number of processors on a P-RAM.

*Proof.* We need a result of Dahlhaus *et al.* In [7], it was shown that if an  $n$ -node graph has minimum degree at least  $\frac{n}{2}$ , then a Hamiltonian path in the graph can be found in time  $O(\log^4 n)$  using polynomial number of processors on a P-RAM. To find  $P$ , we may first compute a Hamiltonian path  $p$  in  $G$  by using the NC<sup>4</sup> algorithm above, and then compute  $\frac{|p|}{f(n)+1}$  vertex disjoint paths of length  $f(n)$  from  $p$  and put them into  $P$ . ■

By combining Theorem 4.1 and Lemma 4.3, we immediately have the following corollary.

**Corollary 4.2** Given an  $n$ -node graph with minimum degree at least  $\frac{n}{2}$ , a maximal  $f(n)$ -diameter tree cover of the graph can be found in time  $O(\log^4 n)$  using polynomial number of processors on a P-RAM.

## 5 The second algorithm

In the last section, we gave an algorithm for the MDTC( $f$ ) problem and proved that the algorithm has NC implementations if the magnitude of  $f$  is restricted to a small number. In this section, we give another algorithm for the MDTC( $f$ ) problem and show that it has an NC implementation if the magnitude of  $f$  is restricted to a rather large number. This algorithm proceeds in stages. In each stage, a portion of a maximal  $f(n)$ -diameter tree cover is computed and is removed from the input graph. The algorithm halts when the graph becomes empty. Formally, the algorithm has the following description.

### Algorithm 2

*Input:* An  $n$ -node graph  $G_0$ .

*Output:* A maximal  $f(n)$ -diameter tree cover  $C_{out}$ .

*Initialization:* Set  $C_0$  to the empty graph.

*Stage  $i$ :* ( $i \geq 1$ )

1. Set  $G_i$  to the subgraph of  $G_{i-1}$  induced by  $V(G_{i-1}) - V(C_{i-1})$ .

2. If  $G_i$  is empty, then halt and output  $C_{out} = C_0 \cup C_1 \cup \dots \cup C_{i-1}$ .
3. Set  $C_i$  to the empty graph.
4. In parallel, for each connected component  $H$  of  $G_i$ , perform the following steps:
  - 4.1 Compute a spanning tree  $T_H$  of  $H$ ;
  - 4.2 If  $T_H$  contains no path of length  $f(n) + 1$ ,
  - 4.3 then add  $T_H$  to  $C_i$ ,
  - 4.4 else find a maximal set  $P$  of vertex disjoint paths of length  $f(n)$  in  $T_H$  and set  $C_i$  to the output of  $Extend(G_i, P)$ .

End of Algorithm 2.

We now show the correctness of Algorithm 2. In addition to the notations used in the algorithm, let  $m$  be the number of stages required by the algorithm. Then,  $C_{out} = \cup_{1 \leq i \leq m} C_i$ .

**Lemma 5.1**  $C_{out}$  is a maximal  $f(n)$ -diameter tree cover of  $G_0$ .

*Proof.* By the algorithm and Theorem 3.1, it is easy to see that  $C_i$  contains neither a path of length  $f(n) + 1$  nor a cycle. Noting that  $C_i$  and  $C_j$  share no common vertex when  $i \neq j$ , we have that  $C_{out}$  is an  $f(n)$ -diameter tree cover of  $G_0$ . Next, we show that for every edge  $e \in E(G_0) - E(C_{out})$ , adding  $e$  to  $C_{out}$  yields either a path of length  $f(n) + 1$  or a cycle. Let  $e = \{w_1, w_2\}$  be an arbitrary edge in  $E(G_0) - E(C_{out})$ . Then, there are two cases that can occur:

*Case 1:* both  $w_1$  and  $w_2$  are contained in  $C_i$  for some  $i$  with  $1 \leq i \leq m$ . By the algorithm and Theorem 3.1, we immediately have that adding  $e$  to  $C_{out}$  yields either a path of length  $f(n) + 1$  or a cycle in  $C_{out}$ .

*Case 2:*  $w_1$  is contained in  $C_i$  and  $w_2$  is contained in  $C_j$  with  $i \neq j$ . W.l.o.g., we may assume that  $i < j$ . Let  $T$  be the connected component of  $C_i$  that contains  $w_1$ . If  $T$  is a spanning tree of some connected component of  $G_i$ , then  $w_2$  could have been in  $C_i$  by the algorithm and then we have a contradiction. So we may assume that  $T$  is not a spanning tree of a connected component of  $G_i$ . Then by the algorithm,  $T$  must be obtained by using procedure *Extend*. Now Theorem 3.1 shows that adding  $e$  to  $C_{out}$  yields a path of length  $f(n) + 1$  in  $C_{out}$ . ■

We next give the running time of Algorithm 2.

**Lemma 5.3** Algorithm 2 can be implemented in time  $O(\frac{n \log^2 n}{f(n)+1})$  using polynomial number of processors on a P-RAM.

*Proof.* It is easy to see that each individual step of Algorithm 2 can be performed in  $O(\log^2 n)$  time using polynomial number of processors on a P-RAM. Since the number of vertices in  $G_i$  is less than that in  $G_{i-1}$  at least  $f(n) + 1$  for  $2 \leq i \leq m$ , the number of stages required by Algorithm 2 is no more than  $\frac{n}{f(n)+1}$ . Hence the lemma follows. ■

The following theorem summarizes the results above.

**Theorem 5.1.** For every function  $f$ , Algorithm 2 outputs a maximal  $f(n)$ -diameter tree cover of a given  $n$ -node, and can be performed in  $O(\frac{n \log^2 n}{f(n)+1})$  time using polynomial number of processors on a P-RAM.

The following corollary follows immediately from the above theorem.

**Corollary 5.1** A maximal  $f(n)$ -diameter tree cover of a given  $n$ -node graph can be found in  $O(\log^{k+2} n)$  time using polynomial number of processors on a P-RAM if  $f(n) = \Omega(\frac{n}{\log^k n})$ .



## 6 The third algorithm

In this section, we present our third algorithm for the MDTC( $f$ ) problem. The key idea used in the algorithm is that of *path separators* [3, 4]. A *path separator* of an  $n$ -node connected graph  $G$  is a path  $p$  in  $G$  such that when all vertices on  $p$  are removed from  $G$ , the resulting graph has no connected component with more than  $\frac{n}{2}$  vertices. The following lemma is an immediate consequence of Aggarwal *et al.*'s results [4].

**Lemma 6.1** [4]. There exists an RNC<sup>5</sup> algorithm which, given a connected graph  $G$ , finds a path separator of  $G$ .

Next we give the description of our algorithm for the MDTC( $f$ ) problem. This algorithm proceeds in stages. In each stage, a portion of a maximal  $l$ -diameter tree cover is computed and is removed from the input graph. The algorithm halts when the graph becomes empty. Since we use path separators to halve the graph in size in each stage, the number of stages required is  $O(\log n)$ . Formally, the algorithm has the following description.

**Algorithm** *Find\_Max\_Tree\_Cover*( $G, l$ )

*Input:* A graph  $G_0 = (V_0, E_0)$  and a positive integer  $l$ .

*Output:* A maximal  $l$ -diameter tree cover  $C_{out}$  of  $G_0$ .

*Initialization:* Set  $C_0$  to the empty graph.

**begin**

*Stage  $i$ : ( $i \geq 1$ )*

1. Set  $G_i$  to the subgraph of  $G_{i-1}$  induced by  $V(G_{i-1}) - V(C_{i-1})$ .
2. If  $G_i$  is empty, then halt and output  $C_{out} = \cup_{0 \leq j \leq i-1} C_j$ .
3. Set  $C_i$  to the empty graph.
4. In parallel, for each connected component  $H$  of  $G_i$ , perform the following steps:
  - 4.1 Compute a path separator  $p$  of  $H$ .
  - 4.2 Divide  $p$  as  $p_1, e_1, p_2, \dots, e_{k-1}, p_k$  such that  $|p_i| = l$  for  $1 \leq i \leq k-1$ ,  $|p_k| \leq l$ , and  $e_i$  is the edge on  $p$  between the end vertex of  $p_i$  and the start vertex of  $p_{i+1}$ .
  - 4.3 If  $|p_k| = l$ , then add the output of  $Extend(H, \{p_1, \dots, p_k\})$  to  $C_i$  and go to Stage  $i+1$ .
  - 4.4 If the output of  $Extend(H, \{p_1, \dots, p_{k-1}\})$  contains all vertices in  $V(p_k)$ , then add the output of  $Extend(H, \{p_1, \dots, p_{k-1}\})$  to  $C_i$  and go to Stage  $i+1$ .
  - 4.5 Set  $H'$  to the subgraph of  $H$  induced by  $V(H) - V(p_k)$ .
  - 4.6 Add the output  $H''$  of  $Extend(H', \{p_1, \dots, p_{k-1}\})$  to  $C_i$ .
  - 4.7 Set  $H'''$  to the connected component of the subgraph of  $H$  induced by  $V(H) - V(H'')$  such that  $H'''$  contains  $p_k$ .
  - 4.8 Compute a spanning tree  $T$  of  $H'''$  such that  $T$  contains  $p_k$ .
  - 4.9 If  $T$  contains no path of length  $l+1$ , then add  $T$  to  $C_i$  and go to Stage  $i+1$ .
  - 4.10 Find a path  $p'$  of length  $l$  in  $T$  such that  $\max\{dist_T(v_1, u), dist_T(v_2, u)\} \leq l$  for each  $u \in V(p_k)$ , where  $v_1$  and  $v_2$  are  $p'$ 's endpoints.
  - 4.11 Add the output of  $Extend(H''', p')$  to  $C_i$ .

**End**

We below show the correctness and the running time of the algorithm. In addition to the notations used in the algorithm, let  $m$  be the number of stages required by the algorithm. Then,  $C_{out} = \cup_{0 \leq i \leq m} C_i$ .

**Lemma 6.2**  $C_{out}$  is a maximal  $l$ -diameter tree cover of  $G_0$ .

*Proof.* It is easy to see that  $V(C_{out}) = V_0$  and that the induced graph of each  $C_i$  contains no path with more than  $l$  edges nor a cycle by the algorithm and Theorem 3.1. Noting that  $C_i$  and  $C_j$  share no common vertex when  $i \neq j$ , we have that  $C_{out}$  is an  $l$ -diameter tree cover of  $G_0$ . Next, we show that for each  $e \in E_0 - E(C_{out})$ , the graph  $(V_0, E(C_{out}) \cup \{e\})$  contains a path with more than  $l$  edges or a cycle. To show this, it suffices to show that for each  $i$  with  $1 \leq i \leq m$  and each edge  $\{w_1, w_2\} \in E(G_i) - E(C_i)$  with  $w_1 \in V(C_i)$  or  $w_2 \in V(C_i)$ , the graph  $(V(C_i) \cup \{w_1, w_2\}, E(C_i) \cup \{\{w_1, w_2\}\})$  contains a path of length  $l + 1$  or a cycle. This, however, follows immediately from the algorithm, Theorem 3.1, and Fact 3 in Section 3. ■

The following two lemmas show that step 4.8 and step 4.10 can be done in  $NC^2$ .

**Lemma 6.3** Given a connected graph  $G$  with  $n$  vertices and a path  $p$  in  $G$ , a spanning tree  $T$  containing  $p$  can be found in time  $O(\log^2 n)$  using  $O(n^2)$  processors on a P-RAM.

*Proof.* Let  $G = (V, E)$  be a connected graph with  $n$  vertices and let  $p$  be a path in  $G$ . To find a spanning tree  $T$  containing  $p$ , we first introduce a new vertex  $v_{new}$  and construct a graph  $G' = (V', E')$  as follows:

$$V' = (V - V(p)) \cup \{v_{new}\} \text{ and}$$

$$E' = \{\{u, v\} \in E : u, v \notin V(p)\} \cup \{\{v_{new}, v\} : v \notin V(p) \text{ and } (\exists u \in V(p))[\{u, v\} \in E]\}.$$

We then find a spanning tree  $T'$  of  $G'$ . Next we shall describe how to find  $T$  from  $T'$  and  $G$ , by specifying the edge set  $E(T)$  of  $T$ . Initially  $E(T)$  is set to  $E(p)$ . All edges  $\{v, u\} \in E(T')$  with  $v \neq v_{new}$  and  $u \neq v_{new}$  are then added to  $E(T)$ . Finally, for each edge  $\{v_{new}, v\} \in E(T')$ , exactly one edge  $\{u, v\} \in E$  with  $u \in V(p)$  is added to  $E(T)$ . Now, it is easy to see that  $T$  is a spanning tree of  $G$  containing  $p$  and that  $T$  can be found in time  $O(\log^2 n)$  using  $O(n^2)$  processors on a P-RAM. ■

**Lemma 6.4** Let  $(T, p, l)$  be a 3-tuple consisting of an  $n$ -node tree  $T$ , a path  $p$  in  $T$ , and a positive integer  $l$  such that  $T$  contains a path of length  $l$  and  $|p| < l$ . Then we can find a path  $p'$  of length  $l$  in  $T$  in time  $O(\log^2 n)$  using  $O(n^2)$  processors on a P-RAM such that  $\max\{dist_T(v_1, u), dist_T(v_2, u)\} \leq l$  for each vertex  $u$  on  $p$ , where  $v_1$  and  $v_2$  are  $p'$ 's endpoints.

*Proof.* It is easy to see that  $T$  must contain such a path  $p'$ . To find such a path  $p'$ , we may check in parallel for each two vertices  $w_1$  and  $w_2$  in  $T$  whether the path from  $w_1$  to  $w_2$  in  $T$  satisfies the condition for  $p'$ . ■

**Lemma 6.5** The number of iterations required by the algorithm is  $O(\log n)$ .

*Proof.* To show the lemma, it suffices to show that for each  $i$  and each connected component  $H$  of  $G_i$ ,  $C_i$  contains all vertices contained in the path separator  $p$  (step 4.1) of  $H$ . By the algorithm and procedure *Extend*, we need only to show that if step 4.10 and step 4.11 of Stage  $i$  are executed, then all vertices in  $V(p_k)$  are contained in the output of  $Extend(H''', p')$ . Let  $v_1$  and  $v_2$  be the endpoints of  $p'$ , and let  $pos(v, p') = \max\{dist_{p'}(v, v_1), dist_{p'}(v, v_2)\}$  for each  $v \in V(p')$ . Fix an arbitrary vertex  $u \in V(p_k)$  to consider. We can first make sure that if  $u \in V(p')$ , then  $u$  is contained in the output of  $Extend(H''', p')$ , by procedure *Extend*. So we may assume that  $u \notin V(p')$ . Then by step 4.10, we know that  $\max\{dist_T(v_1, u), dist_T(v_2, u)\} \leq l$ . Combining this with the fact that  $T$  is a spanning tree of  $H'''$  containing  $p'$ , we have that there must exist a vertex  $v \in V(p')$  such that  $pos(v, p') + dist_{H'''}(v, u) \leq l$ . Now, Fact 3 in Section 3 implies that  $u$  is contained in the output of  $Extend(H''', p')$ . Hence, the lemma follows. ■

By Lemma 6.1 and the results above, we have the following theorem.

**Theorem 6.1** There exists an  $RNC^6$  algorithm for the MDTC( $f$ ) problem.

## 7 Conclusions

In this paper, we have shown that the MDTC( $f$ ) problem can be solved by an NC<sup>2</sup> algorithm if  $f$  maps each positive integer to a fixed constant. It remains open to find an NC algorithm for the MDTC( $f$ ) problem where the magnitude of  $f$  is not bounded to a fixed constant (e.g.,  $f(n) = O(\log n)$ ). We have also shown that the MDTC( $f$ ) problem can be solved by an NC algorithm if the input graph and/or the magnitude of  $f$  are suitably restricted. One obvious question is to loosen these restrictions. Finally, we have shown that the MDTC( $f$ ) problem can be solved by an RNC<sup>6</sup> algorithm. Another obvious question is to design a more efficient RNC algorithm.

## Acknowledgement

The author would like to thank Prof. Takumi Kasai and Dr. Seinosuke Toda for their encouragement and their valuable suggestions.

## References

- [1] N. Alon, L. Babai, and A. Itai, *A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem*, J. Algorithms, 7 (1986), pp. 567-583.
- [2] R. Anderson, *A Parallel Algorithm for the Maximal Path Problem*, Proc. 17th ACM STOC(1985), pp. 33-37.
- [3] A. Aggarwal and R. Anderson, *A Random NC Algorithm for Depth First Search*, Proc. 19th ACM STOC(1987), pp. 325-334.
- [4] A. Aggarwal, R. Anderson, and M. Kao, *Parallel depth-first search in general directed graphs*, SIAM J. Comput. 19 (1990) 397-409.
- [5] R. Anderson and E. Mayr, *Parallelism and the Maximal Path Problem*, Information Processing Letters, 24 (1987), pp. 121-126.
- [6] S. Cook, *A Taxonomy of Problems with Fast Parallel Algorithms*, Inform. and Comput., 64 (1985), pp. 2-22.
- [7] E. Dahlhaus, P. Hajnal, and M. Karpinski, *Optimal Parallel Algorithm for the Hamiltonian Cycle Problem on Dense Graphs*, Proc. 29th ACM FOCS(1988), pp. 186-193.
- [8] S. Even, *Graph Algorithms*, Computer Science Press (1979).
- [9] A. Israeli and A. Itai, *A Fast and Simple Randomized Parallel Algorithm for Maximal Matching*. Computer Science Dept., Technion, Haifa Israel, 1984.
- [10] A. Israeli and Y. Shiloach, *An Improved Maximal Matching Parallel Algorithm*, Tech. Rep. 333, Computer Science Dept., Technion, Haifa Israel, 1984.
- [11] R. Karp and A. Wigderson, *A Fast Parallel Algorithm for the Maximal Independent Set Problem*, Proc. 16th ACM STOC(1984), pp. 266-272.

- [12] F. Lev, *Size Bounds and Parallel Algorithms for Networks*, Report CST-8-80, Dept. of Computer Science, Univ. of Edinburgh (1980).
- [13] M. Luby, *A Simple Parallel Algorithm for the Maximal Independent Set Problem*, Proc. 17th ACM STOC(1985), pp. 1-10.
- [14] S. Miyano, *The lexicographically first maximal subgraph problems: P-completeness and NC algorithms*, Math. Systems Theory 22 (1989), pp. 47-73.
- [15] S. Miyano, *Parallel Complexity and P-complete Problems*, Proc. FGCS'88 (1988), pp. 532-541.
- [16] D. Pearson and V. Vazirani, *A Fast Parallel Algorithm for Finding a Maximal Bipartite Set*, Foundations of Software Technology and Theoretical Computer Science, LNCS 472 (1990), pp. 225-231.
- [17] T. Shoudai and S. Miyano, *Bounded Degree Maximal Subgraph Problems are in NC*, RIFIS Tech. Rep. 27, Kyushu University, 1990.
- [18] T. Shoudai and S. Miyano, *Maximal/Minimal Problems Parallelizable Using the Maximal Independent Set Problem*, paper in preparation, 1991.